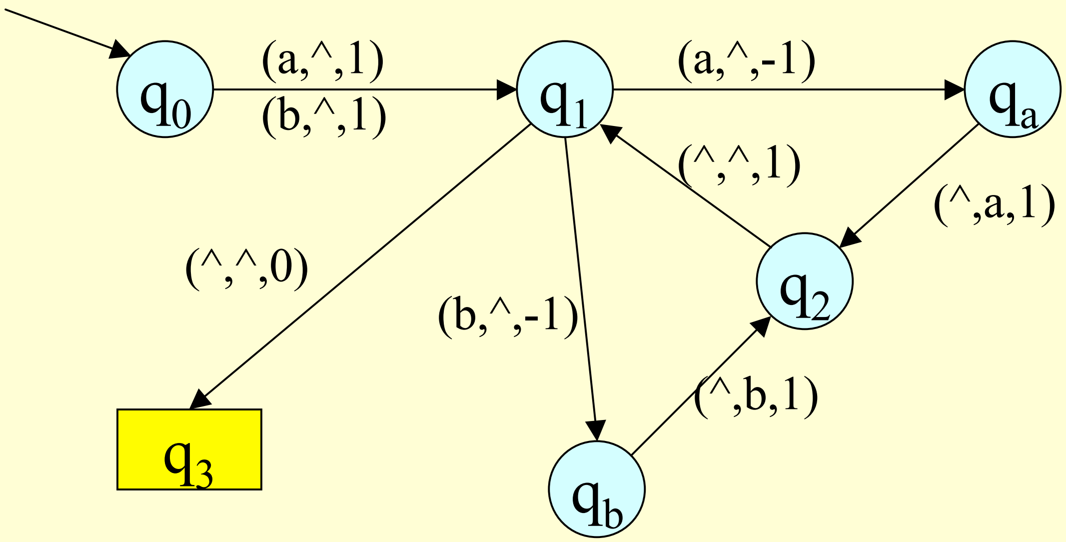
Help for Turing Machine Programmers

First, another variation of the **Tail** function:
overwrites the current symbol with ^ at q_0 and q_1.

Design a TM M* such that for input w, where w = s.w', s ∈ ∑, w' ∈ ∑*,

$f_{M*}(s.w') = w'$ I = {a, b}, ∑ = {a, b, ^}



δ-function

$\delta(q_0,a) = (q_1,\wedge,1)$
$\delta(q_0,b) = (q_1,\wedge,1)$
$\delta(q_1,a) = (q_a,\wedge,-1)$
$\delta(q_1,b) = (q_b,\wedge,-1)$
$\delta(q_a,\wedge) = (q_2,a,1)$
$\delta(q_b,\wedge) = (q_2,b,1)$
$\delta(q_2,\wedge) = (q_1,\wedge,1)$
$\delta(q_1,\wedge) = (q_3,\wedge,0)$

continued..

the **Tail** TM..continued

M* has
- 3-state cycle corresponding to each of {a,b}
- states q_a and q_b for "seen-an-a","seen-a-b"

Limitations of M*?

..if $\sum = \{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p..\}$?

..M* would need a state for "seen-an-α" for each $\alpha \in \sum$.

..but the actions of the TM are the same for each symbol ..

we need the facility to indicate actions for a range of possible symbols, instead of defining separate state(s) to handle each one…

this will allow us to abstract in defining our TM..

BUT we know that a TM will implement this with all the separate states (ok as $\sum$ is finite - so finite number of states required)………

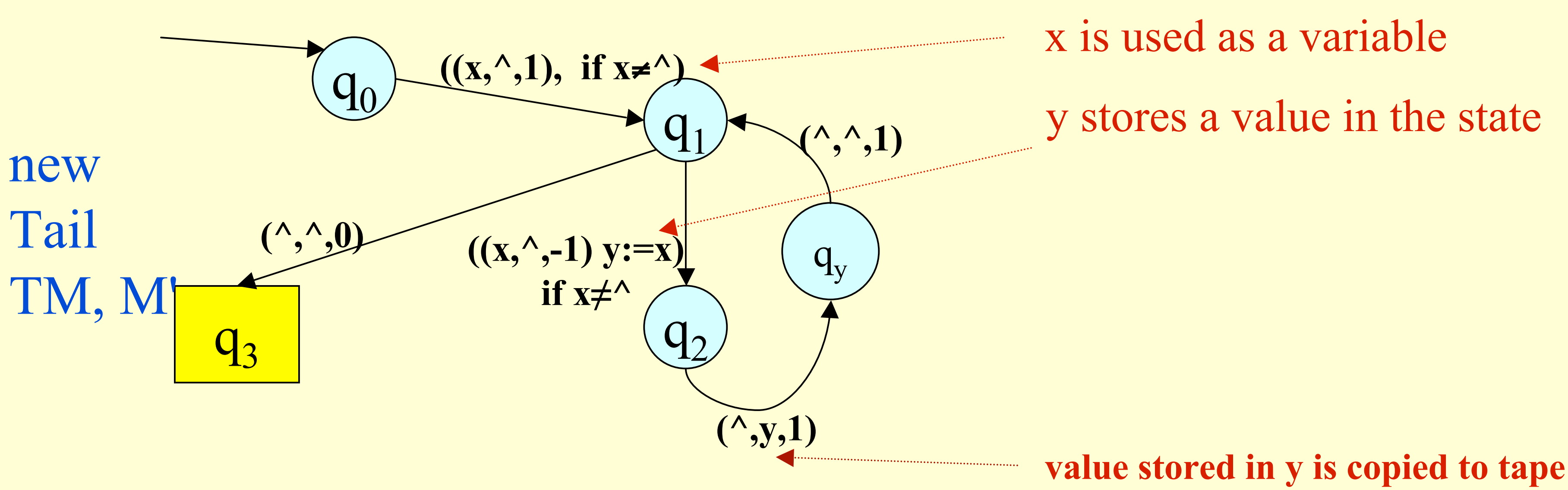
Programmers’ help - 1

Storing a finite amount of information in the state

This is a convenient notation for the programmer, not a new definition

- using variables
  - to represent (finite) sets of symbols
  - to remember a value from a previous state

Any subset of symbols in ∑ must be finite.



Programmers' help 1…continued

Is M' a valid TM?
It is a shorter representation of a TM which has

$Q^I = \{q_0, q_1, q_2, q_3, seena_1, seena_2, \ldots seena_n\}$, all $a_i \in I$ (finite)
$\delta^I : Q^I \times I \cup \{\wedge\}$ into $Q^I \times I \cup \{\wedge\} \times (1, 0, -1\}$, a partial function

$M^I = \{Q^I, I, I \cup \{\wedge\}, q_0, \delta^I, F\}$ which is a valid Turing Machine.

M' is just a shorthand notation for a valid Turing Machine.

Programmers' help 2..Multiple Track Tape

First an example with 2 input values:

- a TM M which takes 2 strings separated by *
  M determines whether the strings are identical.

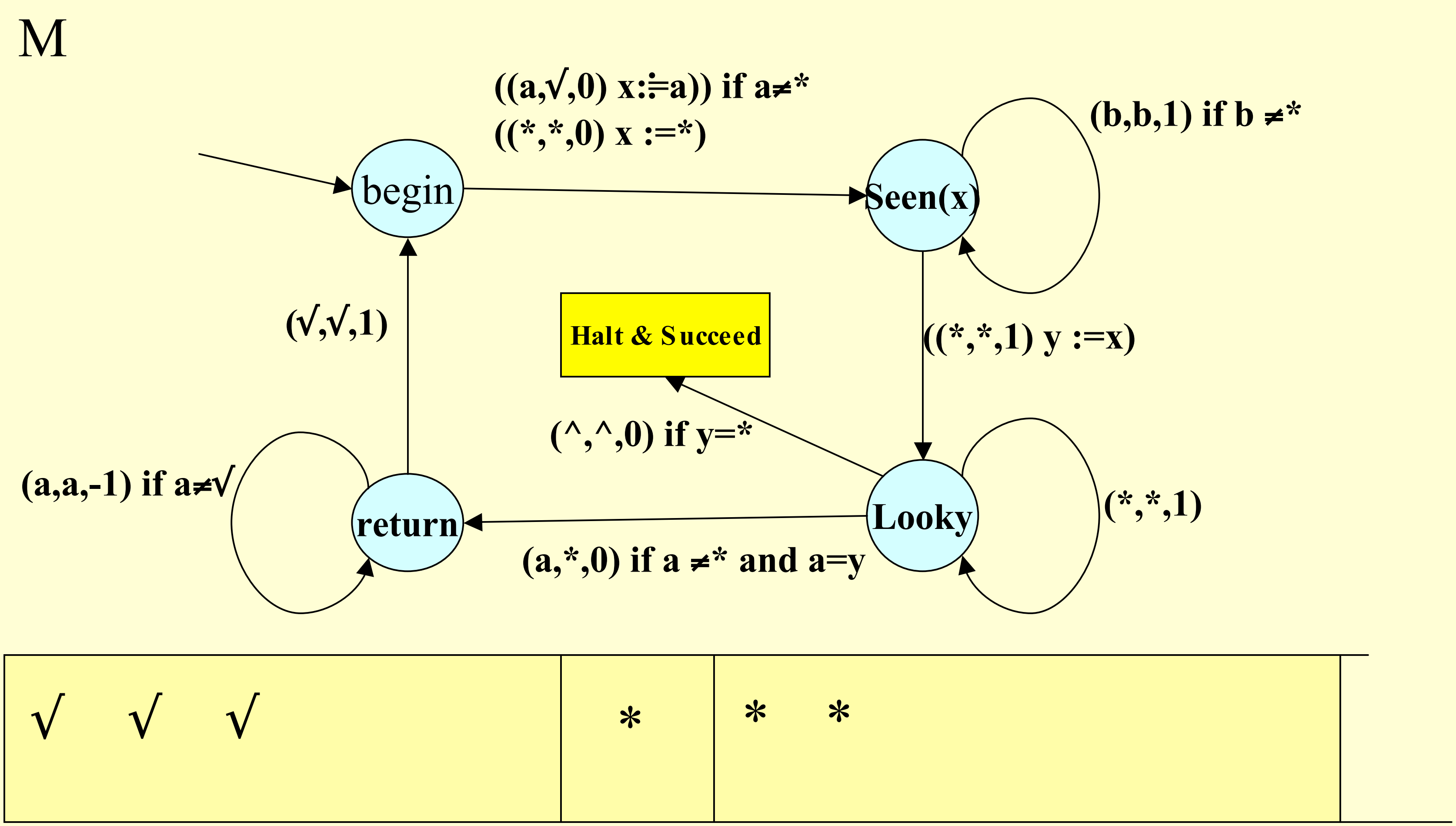
We need just a yes/no result..represented as Halt and Succeed
or Halt and Fail

w_1	*	w_2

M has input alphabet I, full alphabet $\sum = I \cup \{\wedge, \surd\}$

M compares w_1 and w_2 character by character.
It will H & F if any corresponding pair is not equal
H & S if all pairs are equal ie. w_1 and w_2 are identical

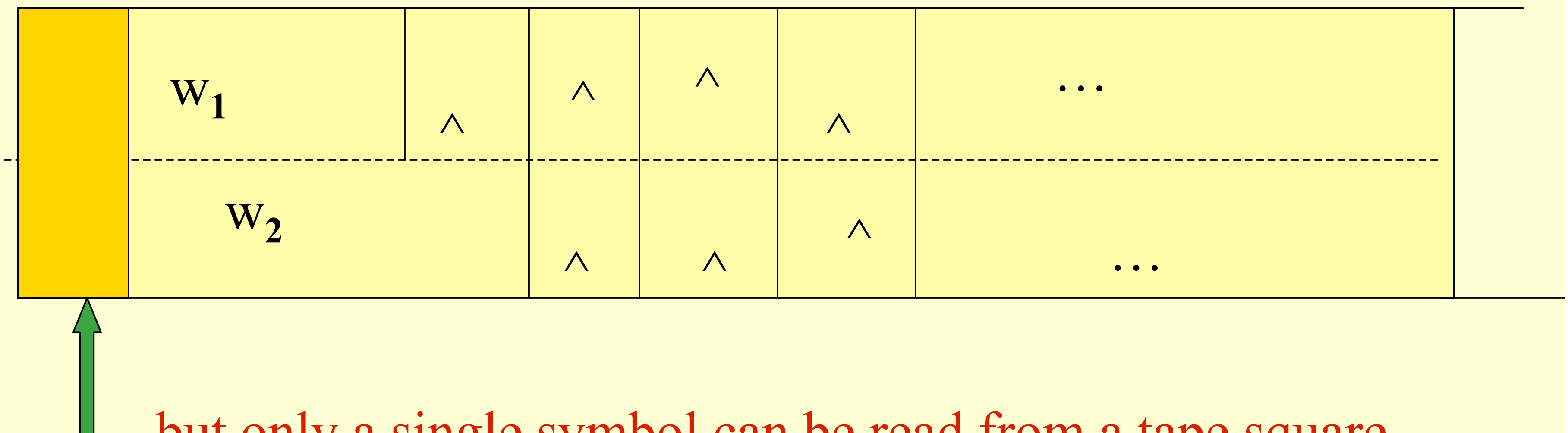
Programmers' help 2..continued



Programmers' help 2..continued

M moved up and down the tape using strings w_1 and w_2 in parallel

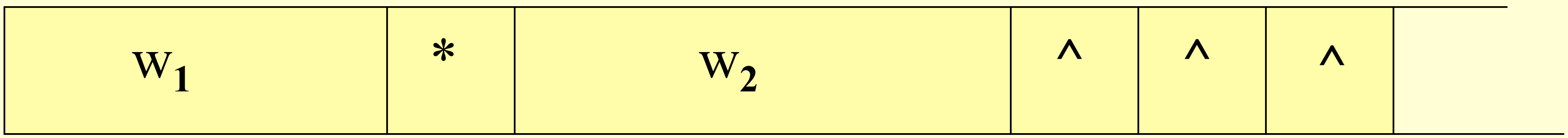
..2-track tape would:
- permit symbols to be compared to be read together
- eliminate the repeated head movement along the tape



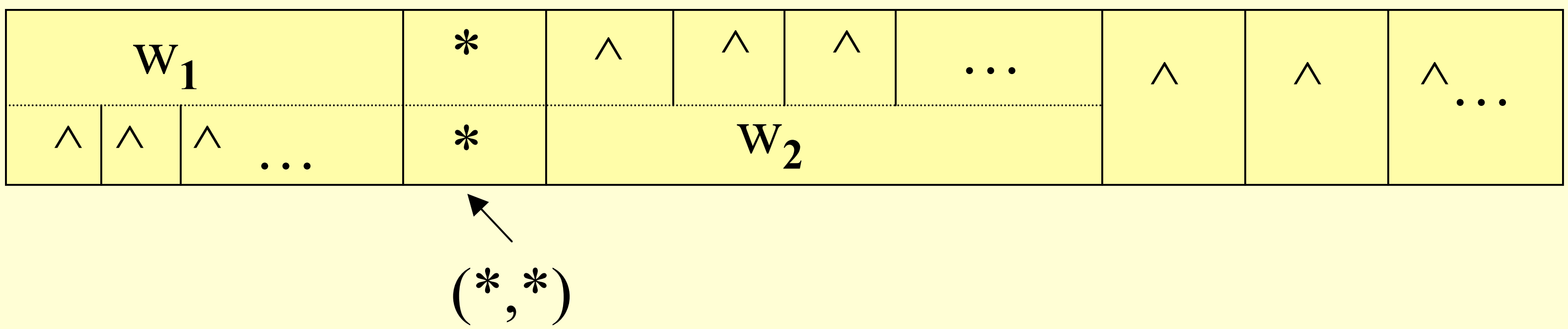
but only a single symbol can be read from a tape square…

the TM to compare w_1 and w_2 for equality..

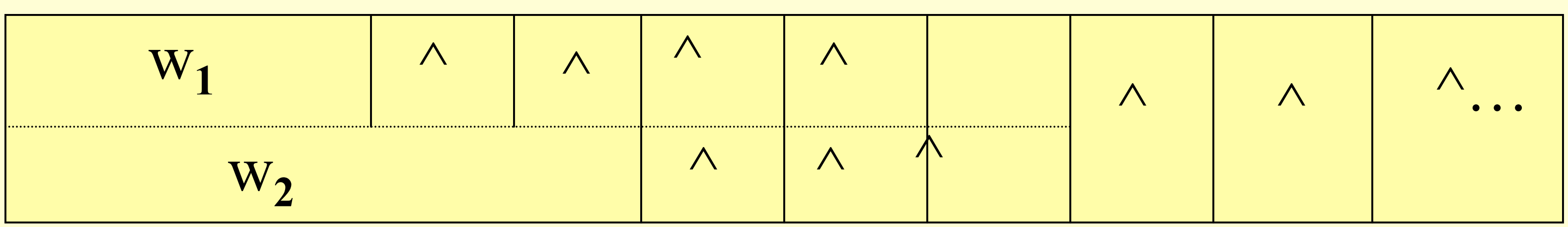
Starting with a single track,



convert to 2-track working using the extended ∑



shift w_2 left to permit reading of w_1 and w_2 together



..so 2-track working is simulated by

- extending the alphabet, ∑, to include new symbols
- representing a pair of symbols by a new symbol

eg. (a_1, a_2), (a_1, a_4)…we interpret the first component as being on the first track..

..extended to n tracks, symbols represent an n-tuple $(a_1, a_2, \dots a_n)$.

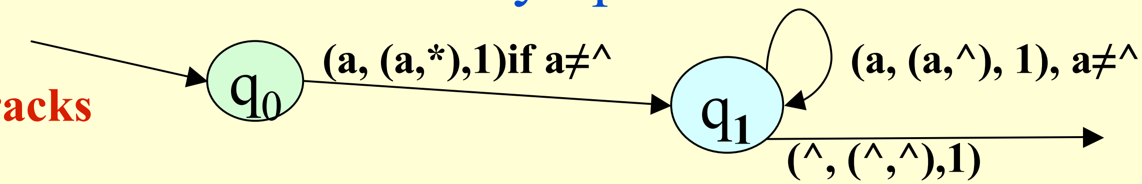
is a TM with n-track tape still a valid TM according to our definition?

- we have extended ∑, but it is still finite.
- the δ-function must be altered for this extended alphabet, but is still a partial function on the state and the symbol read

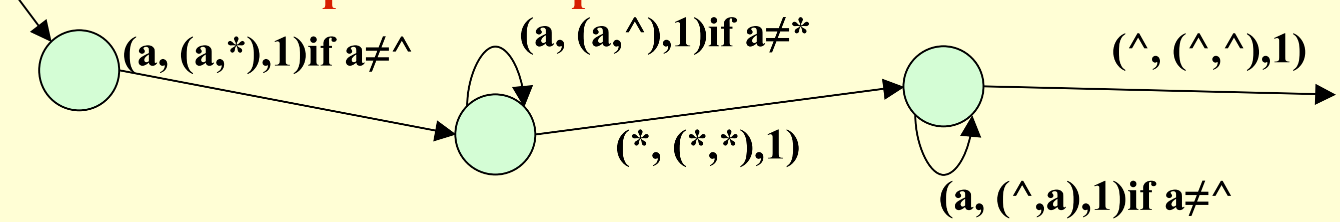
So NO CHANGE to the definition of a Turing Machine.

state diagrams for the necessary operations:

to set up 2 tracks

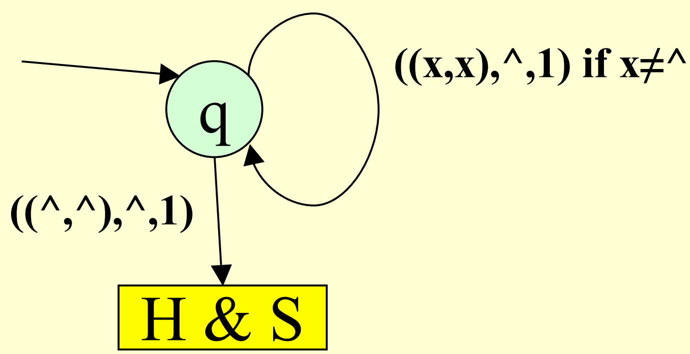


For this example we set up with w1 in track 1 and w2 in track 2: ←



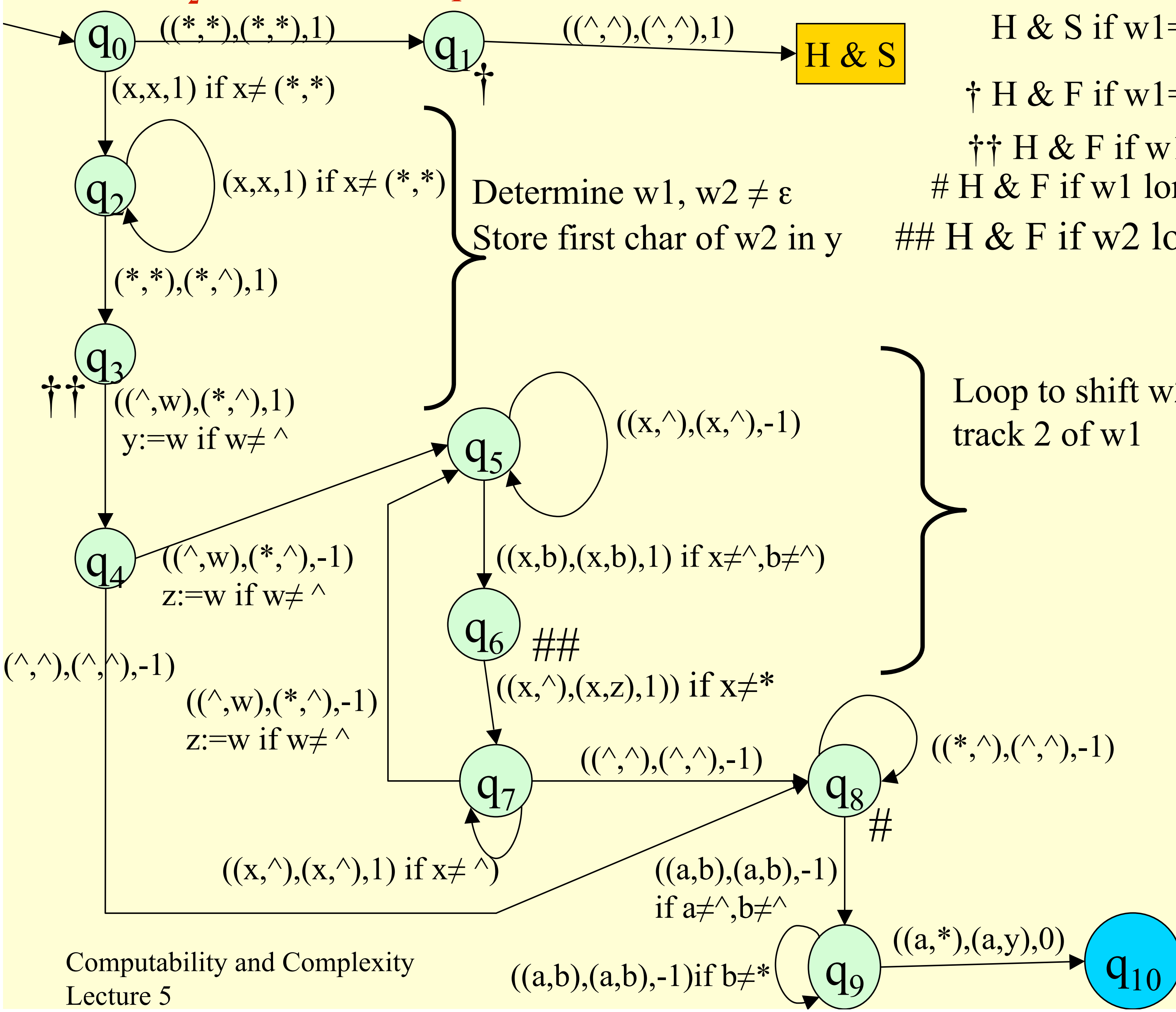
to shift w2 left to start at square 0 (see next slide)..

to read both tracks in the current square, comparing the symbols, starting at square 0



the TM Halts & Succeeds if w1=w2, Halts & Fails otherwise.

to shift w_2 left to start at square 0



H & S if w1= ε and w2= ε

† H & F if w1= ε and w2≠ ε

†† H & F if w1≠ε and w2=ε

H & F if w1 longer than w2

H & F if w2 longer than w1

Determine w1, w2 ≠ ε
Store first char of w2 in y

Loop to shift w2 symbol to track 2 of w1

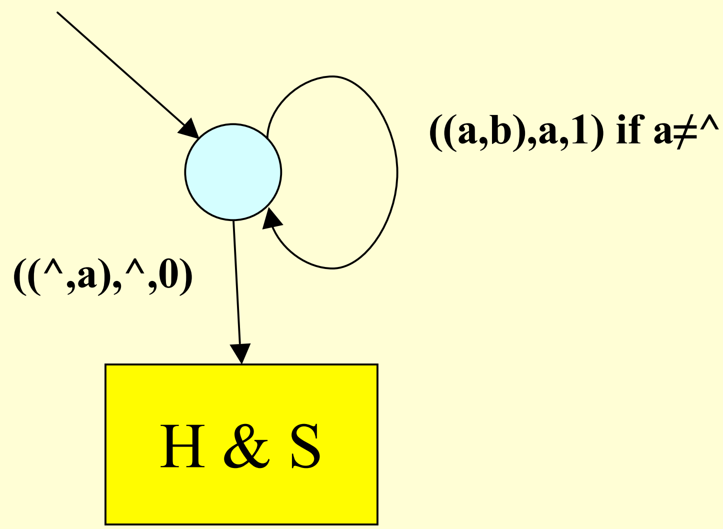
Return to Sq 0 and write y in track 2.

{length(w1)= length(w2)}

output with 2-track tape..

if the TM has output

- the output is built up in the first track, starting at square 0
- the single track must be restored before Halt & Succeed
  starting with head in square 0:



..how do we identify square 0?…

Help for Programmers....identifying square 0..

Why?

to avoid trying to move left from square 0 - causes Halt & Fail

- put a special character in square 0 and shift the input right 1 square.
  shift output word left 1 square before H & S

or

- use a second track..put a special character in it just in square 0
  restore to single track before H & S

Help for Programmers ..3

Turing Machines as subroutines

We can run a series of TMs as a single operation:

• at interchange, convert H & S to initial state for next TM

• all states of all the component TMs + the δ-function form a single large TM.

•ensure tape state and head position are valid at each interchange
(return to square 0?)

Summary

We have seen how to:

- hold finite amounts of data in a state by using a parameter which has a finite number of possible values..

  this is a shorthand notation for the "full" TM which has separate δ-function entries for each symbol and may use different states to 'remember' values.

- simulate multiple tracks on the tape by extending the alphabet, ∑. input and output have a single track as the TM is defined

- identify square 0..one way is to use a second track in square 0

- connect TMs together in a sequence like subroutines
  - (coming next…2-way tape and multiple tapes)